International Research Journal of Engineering and Management Studies (IRJEMS)
Volume: 03 Issue: 03 | March -2019 e-ISSN: 2395-0126

An Algorithm To Detect Separation And Reconnecting Wireless Sensor Network Partitions

R.Jayashree
M.Tech, Department of Computer Science & Engineering, Prist University, Tiruchirapalli, TamilNadu,India.

R.Kalaivani
Assistant Professor, Department of Computer Science & Engineering, Prist University, Tiruchirapalli, TamilNadu,India.

Abstract

Wireless sensor networks (WSNs) are a promising technology for monitoring large regions at high spatial and temporal resolution. The failure of some of its nodes, which is called cut can separate the network into multiple connected components. The ability of detecting cuts by the disconnected nodes and source node of a wireless sensor network will lead to the increase in the operational lifetime of the network. The Distributed Cut Detection (DCD) algorithm proposed here enables every node of a wireless sensor network to detect Disconnected frOm Source events if they occur. Second, it enables a subset of nodes that experience CCOS events to detect them and estimate the approximate location of the cut in the form of a list of active nodes that lie at the boundary of the cut. The algorithm is based on ideas from electrical network theory and parallel iterative solution of linear equations. A key strength of the DCD algorithm is that the convergence rate of the iterative scheme is quite fast and independent of the size and structure of the network.

Index Terms *–Detection and estimation, Iterative computation, Network Separation, Sensor networks, Wireless networks.*

1. Introduction

Wireless sensor networks (WSNs) are a promising technology for monitoring large regions at high spatial and temporal resolution. However, the small size and low cost of the nodes that makes them attractive for widespread deployment also causes the disadvantage of low-operational reliability. A node may fail due to various factors such as mechanical/electrical problems, environmental degradation, battery depletion, or hostile tampering. In fact, node failure is expected to be quite common due to the typically limited energy budget of the nodes that are powered by small batteries. Failure of a set of nodes will reduce the number of multihop paths in the network. Such failures can cause a subset of nodes—that have not failed—to become disconnected from the rest, resulting in a "cut"..

We consider the problem of detecting cuts by the nodes of a wireless network. The source node may be a base station that serves as an interface between the network and its users. Since a cut may or may not separate a node from the source node, we distinguish between two distinct outcomes of a cut for a particular node. When a node u is disconnected from the source, we say that a Disconnected from Source (DOS) event has occurred for u. When a cut occurs in the network that does not separate a node u from the source node, we say that Connected, but a Cut Occurred Somewhere (CCOS) event has occurred for u. By cut detection we mean 1) detection by each node of a DOS event when it occurs, and 2) detection of CCOS events by the nodes close to a cut, and the approximate location of the cut. Nodes that detect the occurrence and approximate locations of the cuts can then alert the source node or the base station. Therefore, on one hand, if a node were able to detect the occurrence of a cut, it could simply wait for the network to be repaired and eventually reconnected, which saves on-board energy of multiple nodes and prolongs their lives. Thus, the ability to detect cuts by both the disconnected nodes and the source node will lead to the increase in the operational lifetime of the network as a whole. A method of repairing a disconnected network by using mobile nodes has been proposed in [1]. Algorithms for detecting cuts, as the one proposed here, can serve as useful tools for such network repairing methods.

In this paper, we propose a distributed algorithm to detect cuts, named the Distributed Cut Detection (DCD) algorithm. The algorithm allows each node to detect DOS events and a subset of nodes to detect CCOS events. The algorithm we propose is distributed and asynchronous: it involves only local communication between neighboring nodes, and is robust to temporary communication failure between node pairs. A key component of the DCD algorithm is a distributed iterative computational step through

which the nodes compute their (fictitious) electrical potentials. The convergence rate of the computation is independent of the size and structure of the network.

2.Related Work

Especially the academic interest in group communication pushed for a partition detection scheme. Babaoglu et al. [3] developed a partitionable group communication service which allows so called "partition-aware applications" to operate in separated network topologies and, after two or more partitions merge, reconfigure themselves. The partitioning problem is handled by a simple *PING/ACK* mechanism. A node sends a *PING* message to another node. If it does not receive an *ACK* in a certain amount of time, that node is added to a list of suspects. A dynamic timeout mechanism is used which leads to a reasonably accurate suspect list. This scheme lacks the ability to distinguish between node failure and partitioning which for most applications is desirable. Also it does not carefully choose the nodes that monitor the network to increase the detection probability.

As noted by Shrivastava *et. al.* [2], the challenges posed by the possibility of network partitioning in WSNs has been recognized in several papers but the problem of detecting when such partitioning occurs seems to have received little attention. To the best of our knowledge, the work by Shrivastava *et. al.* [2] is the only one that addresses the problem of detecting cuts in wireless sensor networks. They developed an algorithm for detecting ǫ linear cuts, which is a linear separation of ǫn nodes from the base station. The reason for the restriction to linear cuts is that their algorithm relies critically on a certain duality between straight line segments and points in 2D, which also restricts the algorithm in [2] to sensor networks deployed in the 2D plane. The algorithm developed in [2] needs a few nodes called sentinels that communicate with a base station either directly or through multi-hop paths. The base station detects -cuts by monitoring whether it can receive messages from the sentinels.

In contrast to the algorithm in [2], the DSSD algorithm proposed in [6] is not limited to -linear cuts; it can detect cuts that separate the network into multiple components of arbitrary shapes. Furthermore, the DSSD algorithm is not restricted to networks deployed in 2D, it does not require deploying sentinel nodes, and it allows every node to detect if a cut occurs.The DSSD algorithm involves only nearest neighbor communication, which eliminates the need of routing messages to the source node. This feature makes the algorithm applicable to mobile nodes as well. Since the computation that a node has to carry out involves only averaging, it is particularly well suited to wireless sensor networks with nodes that have limited computational capability.In this paper,the proposed algorithm is an extension of previous work [6], which partially examined the DOS detection problem.

3.Distributed Cut Detection

3.1 Definition and Problem Statement

Time is measured with a discrete counter k=-∞,...,-1,0,1,2,...We model a sensor network as a time-varying graph = ($\mathcal{E}$) whose node set V(k) represents the sensor nodes active at time k and the edge set $\mathcal{E}$( ) consists of pairs of nodes (u,v) such that nodes u and v can directly exchange messages between each other at time k. By an active node we mean a node that has not failed permanently. All graphs considered here are undirected, i.e., (,)=(,). The neighbors of a node is the set of nodes connected to , i.e. = { |(,) ∈ $\mathcal{E}$}. The number of neighbors of ,| |,is called its degree, which is denoted by d_i. A path from to is a sequence of edges connecting and . A graph is called connected if there is a path between every pair of nodes. A component of a graph is a maximal connected subgraph of (i.e., no other connected subgraph of contains as its subgraph).

In terms of these definitions, a cut event is formally defined as the increase of the number of components of a graph due to the failure of a subset of nodes (as depicted in Fig. 1). The number of cuts associated with a cut event is the increase in the number of components after the event.

The problem we seek to address is twofold. First, we want to enable every node to detect if it is disconnected from the source (i.e., if a DOS event has occurred). Second, we want to enable nodes that lie close to the cuts but are still connected to the source (i.e., those that experience CCOS events) to detect CCOS events and alert the source node.

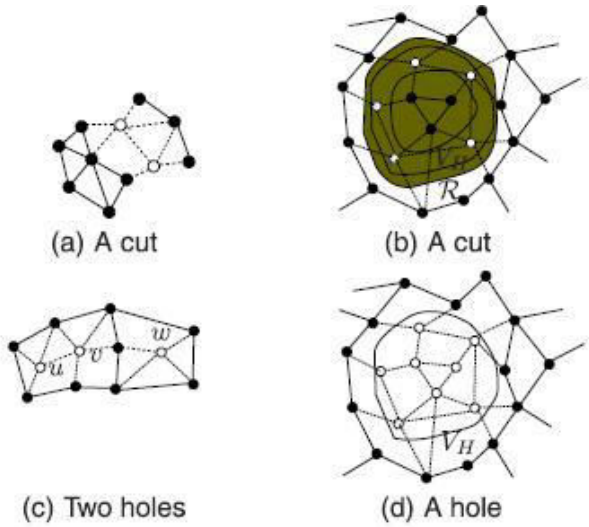


Fig 1.Examples of cuts and holes. Filled circles represent active nodes and unfilled filled circles represent failed

nodes. Solid lines represent edges, and dashed lines represent edges that existed before the failure of the nodes. The hole in (d) is indistinguishable from the cut in (b) to nodes that lie outside the region R.

There is an algorithm-independent limit to how accurately cuts can be detected by nodes still connected to the source, which are related to holes. Fig. 1 provides a motivating example. We allow the possibility that the algorithm may not be able to tell a large hole (one whose circumference is larger than ℓmax) from a cut, since the examples of Figs. 1b and 1c show that it may be impossible to distinguish between them. Note that the discussion on hole detection part is limited to networks with nodes deployed in 2D.

3.2 State Update Law and Electrical analogy

The DCD algorithm is based on the following electrical analogy. Imagine the wireless sensor network as an electrical circuit where current is injected at the source node and extracted out of a common fictitious node that is connected to every node of the sensor network. Each edge is replaced by a 1 Ω resistor. When a cut separates certain nodes from the source node, the potential of each of those nodes becomes 0, since there is no current injection into their component. The potentials are computed by an iterative scheme (described in the sequel) which only requires periodic communication among neighboring nodes. The nodes use the computed potentials to detect if DOS events have occurred (i.e., if they are disconnected from the source node).

To detect CCOS events, the algorithm uses the fact that the potentials of the nodes that are connected to the source node also change after the cut. Therefore, CCOS detection proceeds by using probe messages that are initiated by certain nodes that encounter failed neighbors, and are forwarded from one node to another in a way that if a short path exists around a "hole" created by node failures, the message will reach the initiating node. The nodes that detect CCOS events then alert the source node about the cut.

Every node keeps a scalar variable, which is called its state. The state of node at time k is denoted by ( ). Every node initializes its state to 0, i.e., 0 = 0 ,∀ .During the time interval between the kth and k + 1th iterations, every node broadcasts its current state ( ) and listens for broadcasts from its current neighbors. Let ( ) be the set of neighbors of node at time k. Assuming successful reception, has access to the states of its neighbors, i.e., for j∈ ( ) , at the end of this time period. The node then updates its state according to the following state update law (the index = 1 corresponds to the source node), where the source strength s (a positive number) is a design parameter:

where ≔ | | is the degree of node at time k, and 1 ( ) is the indicator function of the set A. That is, 1 $_1$ = 1if = 1 (sourcnode)

, 1 $_1$ = 0 if ≠ 1 and . After the state is updated, the next iteration starts. At deployment, nodes go through a neighbor discovery and every node determines its initial neighbor set (0). After that, can update its neighbor list ( ) as follows: If no messages have been received from a neighboring node for the past iterations, node drops that node from its list of neighbors. The integer parameter is a design choice.

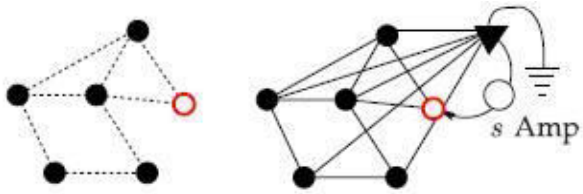
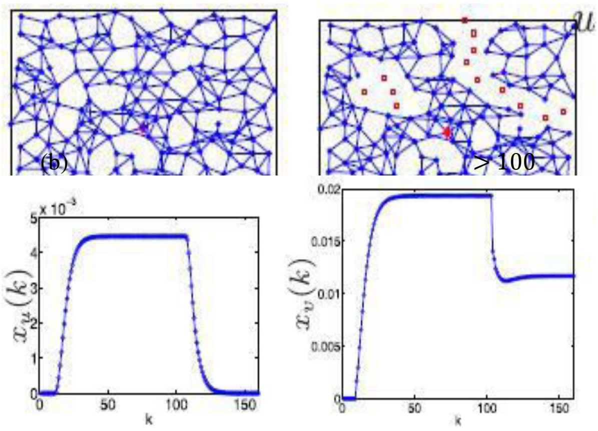


Fig. 2. A graph describing a sensor network (left), and the associated fictitious electrical network ^{elec}(right). s Amp current is injected into the electrical network through the "source node" (unfilled circle), and extracted through the "ground" node (filled triangle). The line segments in the electrical network are 1Ω resistors.

To understand the state update law's relation to the electrical analogy described earlier, given an undirected graph (, $\mathcal{E}$), imagine a fictitious graph ^{elec}=(^{elec}, $\mathcal{E}^{elec}$) as follows: The node set of the fictitious graph is ^{elec} = ∪ { }, where g is a fictitious grounded node; and every node in is connected to the grounded node g with a single edge, which constitute the extra edges in $\mathcal{E}^{elec}$ that are not there in $\mathcal{E}$. Now an electrical network (^{elec} ,1) is imagined by assigning to every edge of ^{elec} a resistance of 1Ω . Fig. 2 shows a graph and the corresponding fictitious electrical network (^{elec},1). The state update law is simply an iterative procedure to compute the node potentials in the electrical network (^{elec} ,1) in which s Ampere current is injected at the source node and extracted through the grounded node . The potential of the grounded node is held at 0.

When the sensor network is connected, the state of a nodeconverges to its potential in the electrical network (^{elec} ,1),which is a positive number. If a cut occurs, the potential of anode that is disconnected from the source is 0; and this is the value its state converges to. If reconnection occurs after a cut, the states of reconnected nodes again converge to positive values. Therefore, a node can monitor whether it is connected or separated from the source by examining its state. The above description

assumes that all updates are done synchronously. In practice, especially with wireless communication, an asynchronous update is preferable. The algorithm can be easily extended to asynchronous setting by letting every node keep a buffer of the last received states of its neighbors. If a node does not receive messages from a neighbor during the interval between two iterations, it updates its state using the last successfully received state from that neighbor. In the asynchronous setting every node keeps a local iteration counter that may differ from those of other nodes by arbitrary amount.



(c) state of node (d) state of node

Fig. 3 shows the evolution of the node states in a network of 200 nodes when the states are computed using the update law described above.

The source node is at the center. The nodes shown as red squares in Fig. 3b fail at k = 100, and thereafter they do not participate in communication or computation. Figs. 3c and 3d show the time evolution of the states of the two nodes and , which are marked by circles in Fig. 3b. The state of node (that is disconnected from the source due to the cut) decays to 0 after reaching a positive value, whereas the state of the node (which is still connected after the cut) stays positive.

4.Distributed Cut Detection Algorithm

4.1 DOS Detection

When a node u is disconnected from the source, we say that a Disconnected from Source (DOS) event has occurred for u. The algorithm allows each node to detect DOS events. The nodes use the computed potentials to detect if DOS events have occurred (i.e., if they are disconnected from the source node). The approach here is to exploit the fact that if the state is close to 0 then the node is disconnected from the source, otherwise not. In order to reduce sensitivity of the algorithm to variations in network size and structure, we use a normalized state. DOS detection part consists of steady-state detection, normalized state computation, and connection/separation detection.A node keeps track of the positive steady states seen in the past using the following method. Each node computes the normalized state difference ( ) as follows:

$$\delta x_i(k) = \begin{cases} \dfrac{x_i(k) - x_i(k-1)}{x_i(k-1)}, & \text{if } x_i(k-1) > \epsilon_{\text{zero}}, \\ \infty, & \text{otherwise,} \end{cases} \quad (2)$$

Where zero is a small positive number. A node keeps a Boolean variable Positive Steady State Reached (PSSR) and updates PSSR(k)← 1 if < Δ for = k − Tguard ,k − Tguard +1,…,k(i.e., for Tguard consecutive iterations), where Δ is a small positive number and T_{guard} is a small integer. The initial 0 value of the state is not considered a steady state, so PSSR( )=0 for = 0,1, … , T_{guard} .

Each node keeps an estimate of the most recent "steady state" observed, which is denoted by ( ). This estimate is updated at every time k according to the following rule: if PSSR( )=1, then ← ; otherwise ← − 1 . It is initialized as (0) = ∞.Every node i also keeps a list of steady states seen in the past, one value for each unpunctuated interval of time during which the state was detected to be steady. This information is kept in a vector ( ) , which is initialized to be empty and is updated as follows: If PSSR( ) = 1 but PSSR(− 1) = 0, then is appended to ( ) as a new entry. If steady state reached was detected in both and - 1 (i.e., PSSR( ) = PSSR(− 1) = 1, then the last entry of ( ) is updated to ( ) .

4.2 CCOS Detection

When a cut occurs in the network that does not separate a node u from the source node, we say that Connected, but a Cut Occurred Somewhere (CCOS) event has occurred for u. detection of CCOS events by the nodes close to a cut, and the approximate location of the cut. By "approximate location" of a cut we mean the location of one or more active nodes that lie at the boundary of the cut and that are connected to the source. To detect CCOS events, the algorithm uses the fact that the potentials of the nodes that are connected to the source node also change after the cut. However, a change in a node's potential is not enough to detect CCOS events, since failure of nodes that do not cause a cut also leads to changes in the potentials of their neighbors. Therefore, CCOS detection proceeds by using probe messages.

4.3 Probe Message Indication

Probe messages that are initiated by certain nodes that encounter failed neighbors, and are forwarded from one node to another in a way that if a short path exists around a "hole" created by node failures, the message will reach the initiating node. The pseudo code for the algorithm that decides when to initiate a probe is included. Each PROBE message p contains the following information:

1) A unique probe ID,
2) Source Node id S
3) Destination node,
4) Path traversed (in chronological order), and
5) The angle traversed by the probe message.

The list of probes is the union of the probes it received from its neighbors and the probe it decided to initiate, if any probe is forwarded in a manner such that if the probe is triggered by the creation of a small hole or cut(with circumference less than) the probe traverses a path around the hole in a counter-clockwise (CCW) direction and reaches the node that initiated the probe. Since it is only used to compute destinations of probe messages.

The information required to compute and update these probe variables necessitates the following assumption for CCOS detection:

Assumption 1. 1) The sensor network is a two-dimensional geometric graph, with ^{2} denoting the location of the ith node in a common Cartesian reference frame; 2) Each node knows its own location as well as the locations of its neighbors.

The location information needed by the nodes need not be precise, since it is only used to compute destinations of probe messages. The assumption of the network being 2D is needed to be able to define CW or CCW direction unambiguously, which is used in forwarding probes. At the beginning of an iteration, every node starts with a list of probes to process. The list of probes is the union of the probes it received from its neighbors and the probe it decided to initiate, if any. The manner in which the information in each of the probes in its list is updated by a node.

5.Performance Evaluation

Performance of the DCD algorithm was tested using MATLAB simulations . Two important metrics of performance for the DCD algorithm are 1) detection accuracy, and 2) detection delay. Detection accuracy refers to the ability to detect a cut when it occurs and not declaring a cut when none has occurred. DOS detection delay for a node that has undergone a DOS event is the minimum number of iterations (after the node has been disconnected) it takes before the node switches its flag from 0 to

1. CCOS detection delay is the minimum number of iterations it takes after the occurrence of a cut before a node detects it. In detecting disconnection from source (DOS) events, two kinds of inaccuracies are possible. A DOS0/1 error is said to occur if a node concludes it is connected to the source while it is in fact disconnected, i.e., node declares to be 0 while it should be 1. A DOS1/0 error is said to occur if a node concludes that is disconnected from the source while in fact it is connected. In CCOS detection, again two kinds of inaccuracies are possible. A CCOS0/1 error is said to occur when cut (or a large hole) has occurred but not a single node is able to detect it. A CCOS1/0 error is said to occur when a node concludes that there has been a cut (or large hole) at a particular location while no cut has taken place near that location.The algorithm's effectiveness is examined by evaluating the probabilities of the four types of possible errors enumerated above, as well as the detection delays.

6 System Implementation

In this section, we describe the software implementation and evaluation of the DCD algorithm. In software the algorithm was implemented using the Dot Net(c#) language running on windows xp operating system. The system executes in two phases: the Reliable Neighbor Discovery (RND) phase and the DCD Algorithm phase. In the RND phase each node is connected to the source node. Upon receiving the message, the mote updates the number of beacons received from that particular sender. To determine whether a communication link is established, each mote first computes for each of its neighbors the Packet Reception Ratio (PRR), defined as the ratio of the number of successfully received beacons and the total number of beacons sent by a neighbor. A neighbor is deemed reliable if the PRR > 0:8. Next, the DCD algorithm executes. After receiving state information from neighbors, a node updates its state according to (1) in an asynchronous manner and broadcasts its new state. The state is stored in the database

7.Conclusion

The DCD algorithm we propose here enables every node of a wireless sensor network to detect Disconnected from Source events if they occur. Second, it enables a subset of nodes that experience CCOS events to detect them and estimate the approximate location of the cut in the form of a list of active nodes that lie at the boundary of the cut/hole.The DOS and CCOS events are defined with respect to a specially designated source node. The algorithm is based on ideas from electrical network theory and parallel iterative solution of linear

equations.The algorithm works effectively with a large classes of graphs of varying size and structure, without requiring changes in the parameters. For certain scenarios, the algorithm is assured to detect connection and disconnection to the source node without error. A key strength of the DCD algorithm is that the convergence rate of the underlying iterative scheme is quite fast and independent of the size and structure of the network, which makes detection using this algorithm quite fast. Application of the DCD algorithm to detect node separation and reconnection to the source in mobile networks is a topic of ongoing research.

8.References

[1] G. Dini, M. Pelagatti, and I.M. Savino, "An Algorithm for Reconnecting Wireless Sensor Network Partitions," *Proc. European Conf. Wireless Sensor Networks, pp. 253-267,* 2008.

[2] N. Shrivastava, S. Suri, and C. D. T´oth, "Detecting cuts in sensor networks,"in *IPSN '05: Proceedings of the 4th international symposiumon Information processing in sensor networks*, 2005*, pp. 210–217*.

[3] Ö. Babaoglu, R. Davoli, A. Montresor, Group Communication in Partitionable Systems: Specification and Algorithms*, IEEE Transactions on Software Engineering*, 2001*, vol. 27, no.4*.

[4] H. Ritter, R. Winter, and J. Schiller, "A Partition Detection System for Mobile Ad-hoc Networks," Proc. First Ann. *IEEE Comm. Soc. Conf. Sensor and Ad Hoc Comm. and Networks (IEEE SECON '04), pp. 489-497*, Oct. 2004.

[5] M. Hauspie, J. Carle, and D. Simplot, "Partition Detection in Mobile Ad-Hoc Networks," *Proc. Second Mediterranean Workshop Ad-Hoc Networks, pp. 25-27*, 2003.

[6] P. Barooah, "Distributed Cut Detection in Sensor Networks," *Proc. 47th IEEE Conf. Decision and Control, pp. 1097-1102*, Dec.2008.

[7] A.D. Wood, J.A. Stankovic, and S.H. Son, "Jam: A Jammed-Area Mapping Service for Sensor Networks," *Proc. IEEE Real Time Systems Symp*., 2003.